Discrete Applied Mathematics 36 (1992) 233-251
North-Holland

A solvable class of quadratic 0-1 programming*

Srimat T. Chakradhar**

Department of Computer Science and CAIP Research Center, Rutgers University, Piscataway, NJ 08855-1390, USA

Michael L. Bushnell

CAIP Research Center, Rutgers University, Piscataway, NJ 08855-1390, USA

Received 7 December 1988
Revised 29 August 1990

Abstract

Chakradhar, S.T. and M.L. Bushnell, A solvable class of quadratic 0-1 programming, Discrete Applied Mathematics 36 (1992) 233-251.

We show that the minimum of the pseudo-Boolean quadratic function $f(x)=x^{\mathrm{T}}Qx+c^{\mathrm{T}}x$ can be found in linear time when the graph defined by Q is transformable into a combinational circuit of AND, OR, NAND, NOR or NOT logic gates. A novel modeling technique is used to transform the graph defined by Q into a logic circuit. A consistent labeling of the signals in the logic circuit from the set $\{0,1\}$ corresponds to the global minimum of f and the labeling is determined through logic simulation of the circuit. Our approach establishes a direct and constructive relationship between pseudo-Boolean functions and logic circuits.

In the restricted case when all the elements of Q are nonpositive, the minimum of f can be obtained in polynomial time [15]. We show that the problem of finding the minimum of f, even in the special case when all the elements of Q are positive, is NP-complete.

1. Introduction

A pseudo-Boolean quadratic function $f:\{0,1\}^n\to\mathbb{R}$ is defined as $f(x)=x^{\mathrm{T}}Qx+c^{\mathrm{T}}x$, where $Q=[Q_{ij}]$ is an $n\times n$ symmetric matrix of constants with null elements in the diagonal, c is a column vector of n constants, x is a column vector of n binary (0 or 1) variables and x^{T} is the transpose of x. There is no loss of generality due to the null diagonal assumption because $x_i^2=x_i$ for $1\le i\le n$.

* The research reported here was supported by the CAIP Center, Rutgers University, with funds provided by the New Jersey Commission on Science and Technology and by CAIP's industrial members.

** Presently with NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA.

We associate a graph $G_f(\mathcal{V},\mathcal{E})$ ($\mathcal{V}$ is the vertex set and $\mathcal{E}$ the edge set of the graph) with f as follows:

$$\mathcal{V} = \{x_1, x_2, \ldots, x_n\},$$
$$(x_i, x_j) \in \mathcal{E} \quad \Leftrightarrow \quad Q_{ij} \neq 0 \ (i \in \mathcal{V}, j \in \mathcal{V}).$$

If $e \in \mathcal{E}$ is an edge with extremities x_i and x_j we write (x_i, x_j) to denote the edge e. Also, edge (x_i, x_j) is said to be *incident* on vertices x_i and x_j. Many combinatorial optimization problems like the VLSI layout problem, the traveling salesperson problem, the max-cut problem and others can be solved via quadratic 0-1 programming [13,12].

Finding the minimum of f is an NP-complete problem [10]. In this section, we use the phrases *minimum of f* and *global minimum of f*, interchangeably. For the general case, a branch-and-bound algorithm has been proposed by Carter [4] and polynomial algorithms for computing lower bounds of f have been given by Hammer et al. [11]. However, this problem is solvable in polynomial time when all the elements of Q are nonpositive [15]. Barahona [2] showed that if the graph G_f is series-parallel (planar graphs that are not reducible to a fully connected graph of four vertices are series-parallel graphs), then the minimum of f can be obtained in linear time. This result includes, as a special case, the maximum stable set problem in series-parallel graphs that was solved by Boulala and Uhry [3].

In this paper, we present a new class of quadratic 0-1 programming cases solvable in $O(n)$ time where n is the number of variables in the pseudo-Boolean quadratic function. We establish a direct and constructive relationship between quadratic pseudo-Boolean functions and logic circuits. We show that if the graph G_f is transformable into a combinational circuit of *inverters* and 2-input *AND*, *OR*, *NAND* and *NOR logic gates*, then the minimum of f can be obtained in linear time. A novel modeling technique [5] is used to transform G_f into a logic circuit. The minimum of f is determined by identifying the *primary input* signals of the circuit and by performing *logic simulation* for an arbitrary set of 0-1 values for these input signals.

In Section 2, we review logic circuits and logic simulation. Neural modeling as discussed in Section 3 is crucial to transforming G_f into a logic circuit. All instances of the logic simulation problem can be formulated as quadratic 0-1 programs but only a subset of quadratic 0-1 programming cases can be formulated as logic simulation problems. The formulation of the logic simulation problem as a quadratic program gives valuable insights for developing a procedure to isolate the subset of quadratic programs that can be formulated as logic simulation problems. Section 4 formulates the logic simulation problem as a quadratic 0-1 programming problem and Section 5 presents a linear time algorithm to verify whether or not a given quadratic 0-1 program can be formulated as a logic simulation problem. If a formulation exists, the algorithm constructs the logic circuit C corresponding to G_f. Logic simulation of C for an arbitrary set of primary input values yields a minimizing point of the pseudo-Boolean function.

In Section 6 we investigate the complexity of solving restricted versions of the quadratic 0-1 programming problem and prove that the problem of finding the minimum of f, even in the restricted case when all elements of Q are positive, is NP-complete. The proof establishes an interesting connection between quadratic 0-1 programming and test generation for digital circuits.

2. Background on logic circuits

Signals and gates: A *signal* is a Boolean variable that may assume only one of the two possible values represented by the symbols 0 and 1. A *gate* is simply an electronic circuit which operates on one or more input signals to produce an output signal. Typical gates used in logic circuits perform Boolean functions like AND, OR, NAND, NOR and NOT. An AND gate performs the conjunction ($\wedge$) of the input signals. For example, if x_1 and x_2 are the input signals of the AND gate, the output signal x_3 is the conjunction of its two input signals, i.e., $x_3 = x_1 \wedge x_2$. Similarly, an OR gate performs the disjunction ($\vee$) of the input signals. The complement of a Boolean variable is physically realized by a NOT gate, also called an inverter. For example, if x_1 is the input signal to an inverter, the output signal $x_2 = \overline{x_1}$. A NAND gate performs the complement of the conjunction of its input signals. For example, if x_1 and x_2 are the input signals to the NAND gate, then the output signal $x_3 = \overline{x_1 \wedge x_2}$. A NOR gate performs the complement of the disjunction of the input signals. Two other types of gates, *XOR* and *XNOR*, are often used in logic circuits. If x_1 and x_2 are the input signals to the XOR (exclusive-OR) gate, the output signal x_3 is given by $x_3 = x_1 \oplus x_2 = (x_1 \wedge \overline{x_2}) \vee (\overline{x_1} \wedge x_2)$. The exclusive-NOR gate (XNOR) is simply an inversion of XOR and produces $x_3 = \overline{x_1 \oplus x_2}$.

Logic circuits: Logic circuits are modeled as interconnections of gates. For simplicity of presentation, we will only consider gates with one or two input signals and one output signal. We will refer to a particular gate in the circuit by its output signal name. For example, Fig. 1 shows a logic circuit with three gates. The AND gate x_3 has x_1 and x_2 as its input signals. The NOR gate x_4 also has x_1 and x_2 as its input signals and NOR gate x_5 has x_3 and x_4 as input signals. Signals that are not the output signals of any gate in the circuit are called *primary inputs*. For example, signals x_1 and x_2 are the primary inputs of the circuit in Fig. 1. Signals that realize the output functions of the circuit are called *primary outputs*. For example, signal x_5 is the primary output of the circuit in Fig. 1. In general, a circuit can have several primary input and primary output signals.

Logic circuits can be categorized as *combinational* or *sequential*. Consider a logic circuit C consisting of signals $x_1, x_2, \ldots, x_n$. Let $H_C(\mathcal{V}_C, \mathcal{E}_C)$, where $\mathcal{V}_C$ is the vertex set and $\mathcal{E}_C$ the edge set, be the *circuit graph* associated with C. $\mathcal{V}_C = \{x_1, x_2, \ldots, x_n\}$ and there is an arc $(x_i, x_j) \in \mathcal{E}_C$ ($1 \le i \le n$ and $1 \le j \le n$) if there is a gate x_j with x_i as an input signal. If H_C has no directed *cycles* [1], C is a combinational circuit. Otherwise, C is a sequential circuit. For example, it can easily be verified that the

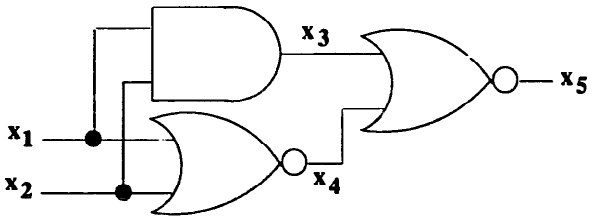

Fig. 1. A combinational circuit example.

circuit graph corresponding to the circuit in Fig. 1 has no directed cycles and therefore, it is a combinational circuit. In the sequel, we will only deal with combinational circuits.

Without loss of generality, let x_n be the primary output of C. The signals in the circuit can always be relabeled to achieve this. We define C^n as the circuit obtained by *deleting* gate x_n from C. More formally, if $H_C(\mathcal{V}_C, \mathcal{E}_C)$ is the circuit graph of C, the circuit graph $H_{C^n}(\mathcal{V}_{C^n}, \mathcal{E}_{C^n})$ of circuit C^n is defined as follows: $\mathcal{V}_{C^n} = \{x_1, x_2, \ldots, x_{n-1}\}$ and there is an arc $(x_i, x_j) \in \mathcal{E}_{C^n}$ if $(x_i, x_j) \in \mathcal{E}_C$ $(1 \le i \le n-1$ and $1 \le j \le n-1)$. For example, x_5 is the primary output of the circuit in Fig. 1 and the circuit obtained by deleting gate x_5 is shown in Fig. 2.

Logic simulation is the process of computing the values of the primary output signals, given a set of values for the primary inputs. For combinational circuits, logic simulation per input value set can be performed in time complexity that is linear in the number of signals in the logic circuit. The first step in logic simulation is *levelizing*. Let $P_i = \{x_j : \text{arc}(x_j, x_i) \in \mathcal{E}_C,\ 1 \le j \le n\}$ be the set of *predecessors* of signal x_i in graph H_C. Therefore, P_i is the set of input signals to gate x_i. We define the *level* of a signal x_i in the combinational circuit C as follows:

- $\text{level}(x_i) = 0$ for all primary input signals,
- $\text{level}(x_i) = \max\{\text{level}(x_j) : x_j \in P_i\} + 1$ for all other signals.

Example 2.1. In Fig. 1, $\text{level}(x_1) = 0$ and $\text{level}(x_2) = 0$ since x_1 and x_2 are primary input signals. Also, $P_3 = P_4 = \{x_1, x_2\}$ and $P_5 = \{x_3, x_4\}$. Therefore, $\text{level}(x_3) = \max\{\text{level}(x_1), \text{level}(x_2)\} + 1 = 1$. Similarly, $\text{level}(x_4) = 1$ and $\text{level}(x_5) = 2$.

The level of every signal in the circuit can be computed in time complexity that

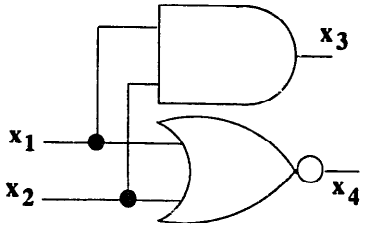

Fig. 2. Logic circuit obtained by deleting gate x_5 from Fig. 1.

is linear in the number of signals in the circuit [9]. Given a set of values for the primary inputs of C, logic simulation is performed as follows:

(1) *Levelize* the circuit. Note that a signal is assigned a higher level number than all its predecessors. As stated earlier, the complexity of levelizing is $O(n)$.

(2) Compute the values of all unknown signals in ascending order of levels. The value of a signal x_i can easily be determined from the signal values of its predecessors. For example, if x_i is the output signal of an AND gate and P_i is the set of input signals to this gate, then x_i is the conjunction of values of signals in P_i. Each signal value can be uniquely determined because when the value of a signal x_i is computed the signal values of all its predecessors have already been computed. Since each gate is visited only once in this process, the complexity of this step is $O(n)$.

Clearly, the work involved in logic simulation is $O(n)$, where n is the number of signals in the circuit.

Example 2.2. Given that the primary input signals x_1 and x_2 are 0 in the circuit in Fig. 1, we will perform logic simulation to compute the values of all other signals. We compute the levels of all signals as shown in Example 2.1 and process the signals in the ascending order of levels: x_3, x_4, x_5 (note that we could have also processed the signals in the order x_4, x_3, x_5 since x_4 and x_3 have the same level number). Since x_3 is the output of an AND gate, $x_3 = x_1 \wedge x_2 = 0 \wedge 0 = 0$. Similarly, $x_4 = \overline{x_1 \vee x_2} = \overline{0 \vee 0} = 1$, and $x_5 = \overline{x_3 \vee x_4} = \overline{0 \vee 1} = 0$.

Consider the AND gate with x_1 and x_2 as its inputs and x_3 as the output. A set of values of x_1, x_2, and x_3 is *consistent* with the function of the AND gate if $x_3 = x_1 \wedge x_2$. Otherwise, the set of values is *inconsistent*. For example, $(x_1 = x_2 = x_3 = 0)$ is consistent with the function of the AND gate but $(x_1 = x_2 = 0, x_3 = 1)$ is inconsistent. An assignment of signal values that is consistent with the function of all gates in the circuit is called a *consistent labeling* of the circuit. For example, signal values determined in Example 2.2 constitute a consistent labeling of the circuit in Fig. 1 since $(x_1 = x_2 = x_3 = 0)$ is consistent with the function of the AND gate x_3, $(x_1 = x_2 = 0, x_4 = 1)$ is consistent with the function of the NOR gate x_4, and $(x_3 = x_5 = 0, x_4 = 1)$ is consistent with the function of the NOR gate x_5. Similarly, performing logic simulation with $x_1 = x_2 = 1$ results in the consistent labeling $(x_1 = x_2 = x_3 = 1, x_4 = x_5 = 0)$. In general, a circuit with p primary inputs has 2^p consistent labelings since performing logic simulation with every set of primary input signal values results in a consistent labeling of the circuit.

3. Logic circuit modeling

We have proposed an unconventional digital circuit modeling technique [7]. Our model represents the function of a digital circuit as an interconnection of computing elements called *neurons* and this approach can be used to obtain efficient solutions for several VLSI problems like logic synthesis and fault detection. We have described

elsewhere the use of these models to solve the fault detection problem in logic circuits [7]. For an understanding of the present discussion, we require some knowledge of neural networks and their use in the modeling of combinational circuits. Familiarity with the modeling technique is essential to understand the transformation of pseudo-Boolean quadratic functions into combinational circuits.

3.1. Neural networks

A *neuron* is a computing element that can assume one of two possible states (also called *activation* values): 0 or 1. A neural network is an interconnection of neurons. A neuron j may be connected to another neuron i by a directed edge or *link* which is characterized by a weight $T_{ij} \in \mathbb{R}$. Each neuron in the network has a *threshold*. Figure 3 shows a typical neuron. $I_i \in \mathbb{R}$ and x_i are the threshold and activation value, respectively, of neuron i.

A neural network is characterized by a pseudo-Boolean quadratic function, called the *energy* function. The neural network is uniquely specified by the weights on the connections between the neurons and by the thresholds of the neurons. For an n-neuron network, the weights are represented as an $n \times n$ matrix $\boldsymbol{T}$ and the thresholds, as an n-component vector $\boldsymbol{I}$. The energy function is of the form [14]:

$$E = -\left[\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}x_i x_j\right] - \left[\sum_{i=1}^{n} I_i x_i\right] + K \tag{1}$$

where $T_{ij} \in \mathbb{R}$ is the weight associated with the connection from neuron j to i, x_i is the activation value of neuron i, $I_i \in \mathbb{R}$ is the threshold of neuron i, and K is a constant. We use Hopfield neural networks [14] in which the connections are bidirectional, i.e., if there is a link from neuron i to neuron j, then there exists a link from j to i. Furthermore, the link weights are symmetric (i.e., $T_{ij} = T_{ji}$, $1 \le i \le n$, $1 \le j \le n$) and $T_{ii} = 0$. We will occasionally refer to neuron i as x_i when there is no confusion between the name of a neuron and its activation value.

$G_E(\mathcal{V}, \mathcal{E})$, the *neural network graph* associated with the energy function E, is defined as follows:

$$\mathcal{V} = \{x_1, x_2, \ldots, x_n\},$$
$$(x_i, x_j) \in \mathcal{E} \quad \Leftrightarrow \quad T_{ij} \neq 0 \quad (i \in \mathcal{V}, j \in \mathcal{V}).$$

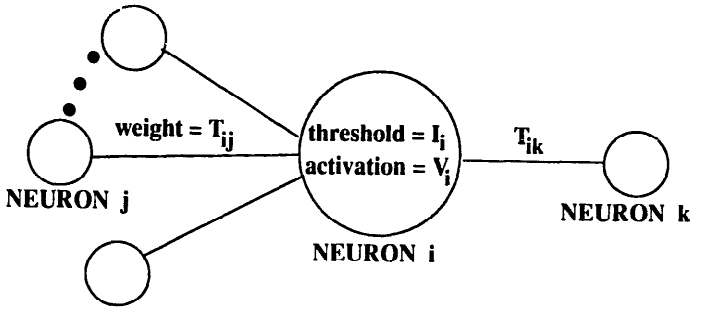

Fig. 3. A typical neuron.

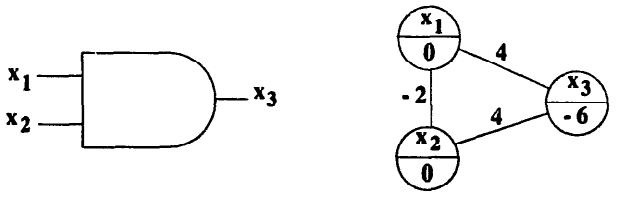

Fig. 4. AND gate and its neural network graph.

Furthermore, we associate a *threshold* $I_i \in \mathbb{R}$ with the vertex $x_i \in \mathcal{V}$ and a *weight* $T_{ij} \in \mathbb{R}$ with the edge $(x_i, x_j) \in \mathcal{E}$.

3.2. *Model for a Boolean gate*

Figure 4 shows an AND gate and the corresponding energy function is given by (this is derived from Table 1, with $A = B = 2$):

$$E_{\text{AND}}(x_3, x_1, x_2) = -[4x_1x_3 + 4x_2x_3 - 2x_1x_2] - [-6x_3]. \tag{2}$$

Variables x_1, x_2, and x_3 can assume only binary values. All operations are arithmetic and not Boolean. It is easily verified that only those values of x_1, x_2, and x_3 that are consistent with the function of the AND gate will satisfy $E_{\text{AND}} = 0$. Furthermore, $E_{\text{AND}} > 0$ for all other combinations of x_1, x_2, and x_3. Note that E_{AND} is a pseudo-Boolean quadratic function that assumes a minimum value of 0 only at values of x_1, x_2, and x_3 consistent with the function of the AND gate. The neural network graph G_{AND} corresponding to equation (2) is shown in Fig. 4. Each signal is represented by a neuron having a threshold. The neurons for signals x_1 and x_2 have 0 thresholds and the neuron for signal x_3 has a threshold of -6. All links are bidirectional and their labels show their weights.

Similar energy functions and the corresponding neural network graphs have been derived [6,7] for all other logic gates. As shown there, it is not important that E_{AND} or the energy function for other gates have a minimum value of 0. In fact, by the addition of a constant, the energy function can be made to have an arbitrary value as its global minimum. Without loss of generality, in the sequel we assume that the energy function for a gate assumes a minimum value of 0. Neural networks for 2-input AND, OR, NAND, NOR, XOR, and XNOR gates and the inverter constitute the *basis set* and gates with more than two inputs are constructed from this basis set. Note that the energy function E_G for a network in the basis set assumes a global minimum value $E_G = 0$ at all consistent network states and $E_G > 0$ for all other network states. Table 1 shows T and I for the neural networks in the basis set in terms of the following functions:

$$\text{inputs}(i,j) = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are input neurons for the same gate,} \\ 0, & \text{otherwise;} \end{cases}$$

$$\text{connected}(i,j) = 1 - \text{inputs}(i,j);$$

$$\delta(i,j)=\begin{cases}1, & \text{if } i=j,\\ 0, & \text{otherwise;}\end{cases}$$

$$\text{output}(i)=\begin{cases}1, & \text{if } i \text{ is an output neuron for the gate,}\\ 0, & \text{otherwise;}\end{cases}$$

$$\text{input}(i)=\begin{cases}1, & \text{if } i \text{ is an input neuron for the gate,}\\ 0, & \text{otherwise.}\end{cases}$$

In Table 1, A, B, and J are constants such that $A>0$, $B>0$, and $J>0$. Neurons corresponding to the primary inputs (outputs) of the circuit are called *primary input (output) neurons*. Neurons corresponding to the input (output) signals of a gate are called *input (output) neurons*. For example, x_1 and x_2 are the input neurons and x_3 is the output neuron in the AND gate neural network graph shown in Fig. 4, and equation (2), the energy function for the AND gate, can be derived from Table 1 by setting $A=B=2$. Energy functions for the XOR and XNOR gates are given elsewhere [7]. From Table 1, clearly, the energy function for a gate in the basis set has at most seven terms and has a minimum value 0 at all consistent states of the gate. Let G_{AND}, G_{OR}, G_{NAND}, G_{NOR}, and G_{NOT} be the neural network graphs corresponding to the energy functions of the AND, OR, NAND, NOR and NOT gates, respectively.

3.3. Circuit modeling with neurons

Consider the three-gate circuit C shown in Fig. 1. The AND gate energy function is given by equation (2). From Table 1, with $A=B=2$, the energy functions for the

Table 1. Basis set of neural networks.

Gate	T and I
AND	$T_{ij}=(1-\delta(i,j))\times((A+B)\times\text{connected}(i,j)-B\times\text{inputs}(i,j))$ $I_i=-(2A+B)\times\text{output}(i)$ $K=0$
OR	$T_{ij}=(1-\delta(i,j))\times((A+B)\times\text{connected}(i,j)-B\times\text{inputs}(i,j))$ $I_i=-B\times\text{output}(i)-A\times\text{input}(i)$ $K=0$
NAND	$T_{ij}=(1-\delta(i,j))\times((-A-B)\times\text{connected}(i,j)-B\times\text{inputs}(i,j))$ $I_i=(2A+B)\times\text{output}(i)+(A+B)\times\text{input}(i)$ $K=2A+B$
NOR	$T_{ij}=(1-\delta(i,j))\times((-A-B)\times\text{connected}(i,j)-B\times\text{inputs}(i,j))$ $I_i=B$ $K=B$
NOT	$T_{ij}=-2J\times(1-\delta(i,j))$ $I_i=J$ $K=J$

two NOR gates are:

$$E_{\mathrm{NOR}}(x_4, x_1, x_2) = 4x_1x_4 + 4x_2x_4 + 2x_1x_2 - 2(x_1 + x_2 + x_4) + 2,$$
$$E_{\mathrm{NOR}}(x_5, x_3, x_4) = 4x_3x_5 + 4x_4x_5 + 2x_3x_4 - 2(x_3 + x_4 + x_5) + 2.$$

The energy function for the entire circuit C is obtained by summing the individual gate energy functions. Thus,

$$E_C = E_{\mathrm{AND}}(x_3, x_1, x_2) + E_{\mathrm{NOR}}(x_4, x_1, x_2) + E_{\mathrm{NOR}}(x_5, x_3, x_4). \tag{3}$$

This simple procedure can model circuits with any connectivity and any types of gates [7,5]. Since the models are defined only for 2-input gates, gates with larger fan-in are represented as combinations of 2-input gates. Individual gate energy functions only assume nonnegative values and, therefore,

- E_C is nonnegative,
- $E_C = 0$ only when the individual energy functions separately become 0,
- any solution of $E_C = 0$ is a consistent labeling for the entire circuit.

Since a logic circuit with p primary inputs has 2^p consistent labelings, the corresponding energy function has 2^p minimizing points. Also, if g is the number of gates in the logic circuit, the energy function for the circuit has at most $7 \times g$ terms since, from Table 1, the energy function for each gate in the logic circuit has at most seven terms. Since $g \leq n$ (n being the number of signals in the circuit), the number of terms in the energy function for the circuit is $O(n)$.

4. Logic simulation as a quadratic 0-1 program

The logic simulation problem in an arbitrary combinational logic circuit C can be reduced to a quadratic 0-1 optimization problem by transforming the logic circuit into an energy function E_C that assumes a minimum value of 0 only at neuron states consistent with the function of all gates in the logic circuit. Logic simulation will require a solution of $E_C = 0$ when the primary input signals assume given values.

Example 4.1. The energy function for the circuit in Fig. 1 is given by equation (3). The logic circuit has two primary inputs and, therefore, only four consistent labelings which are shown in Table 2. The corresponding values of E_C (equation (3)) are shown in the last column. It can be easily verified that E_C assumes the minimum value 0 only at these four consistent labelings of the circuit.

If a pseudo-Boolean quadratic function f can be transformed into a logic

Table 2. Consistent states of circuit in Fig. 1.

x_1	x_2	x_3	x_4	x_5	E_C
0	0	0	1	0	0
0	1	0	0	1	0
1	0	0	0	1	0
1	1	1	0	0	0

circuit, then the minimum of f can be obtained through logic simulation. In the next section, we present a linear time algorithm to transform f into a combinational circuit.

5. Quadratic 0-1 programming as a logic simulation problem

As mentioned earlier, only a subset of quadratic 0-1 programs can be formulated as instances of the logic simulation problem. We now present a linear time algorithm that determines whether or not a given quadratic 0-1 program can be thus formulated. An example is provided to illustrate the mechanics of the algorithm.

5.1. A linear time algorithm

We assume that the function f has the same form as equation (1). Otherwise, f can be rewritten in the same form as equation (1).

Example 5.1. Consider the following quadratic function:

$$f = -5x_1x_3 + 7x_1x_4 + 7x_1x_2 - 5x_2x_3 + 7x_2x_4 + 3x_3x_5 + 3x_4x_5 + 2x_3x_4 - 4x_1 - 4x_2 + 5x_3 - 6x_4 - 2x_5 + 6.$$

It can be rewritten as

$$f = -[5x_1x_2 - 7x_1x_4 - 7x_1x_2 + 5x_2x_3 - 7x_2x_4 - 3x_3x_5 - 3x_4x_5 - 2x_3x_4] - [4x_1 + 4x_2 - 5x_3 + 6x_4 + 2x_5] + 6. \quad (4)$$

The corresponding neural network graph G_f, as defined in Section 3, is shown in Fig. 5. A vertex is indicated by a circle. The vertex label is indicated in the upper half and the vertex weight is indicated in the lower half. For example, vertex x_3 has a vertex weight of -5 since $I_3 = -5$. The graph has five vertices since f is a function of five variables $x_1, \ldots, x_5$. There is an edge between two vertices x_i and x_j if function f has a term $T_{ij}x_ix_j$ and the edge weight T_{ij} is indicated as a label on the edge. For example, edge (x_3, x_4) has an edge weight of -2 since $T_{34} = -2$.

The algorithm attempts to construct a combinational logic circuit from G_f. Each vertex in G_f corresponds to a signal in the logic circuit. The algorithm first deletes vertices and edges of G_f that correspond to logic gates associated with the primary outputs of the logic circuit. Then it recursively constructs the remaining logic gates in the circuit by deleting appropriate edges and vertices in G_f and tracing the signals in the logic circuit from primary outputs to the primary inputs.

We give some graph-theoretic definitions that are used in this section. Let $G(\mathcal{V}, \mathcal{E})$ be a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. If $\mathcal{U} \subseteq \mathcal{V}$, the subgraph induced by $\mathcal{U}$ is the graph $H(\mathcal{U}, \mathcal{F})$ such that $(i,j) \in \mathcal{F}$ if and only if $(i,j) \in \mathcal{E}$ and $\{i,j\} \subseteq \mathcal{U}$. For $v \in \mathcal{V}$, $G \setminus v$ will denote the subgraph induced by the set $\mathcal{V} \setminus v$ ($\setminus$ denotes the set minus operation).

We make the following observations regarding a neural network graph of any combinational circuit. Let C be a circuit with signals $x_1, \ldots, x_n$ and let x_n be a

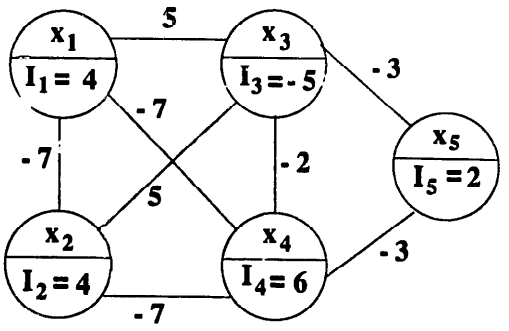

Fig. 5. The graph G_f of function f in Example 5.1.

primary output. Let E_C be the energy function of circuit C and G_C be the associated neural network graph.

Lemma 5.2. *The neural network graph G_C has at least one vertex v of degree one or two. Moreover, if v is of degree two, both edges incident on v will have equal weights.*

Proof. Since x_n is the primary output of circuit C, it is a primary output neuron in G_C. Furthermore, x_n is not an input signal to any gate in the circuit and gate x_n has at most two inputs. Therefore, x_n has degree one (i.e., it is a signal associated with an inverter) or two in G_C. If the degree of x_n is two, then the weights on the edges incident on x_n are equal because the output neuron of a neural network graph of every logic gate in the basis set has two equally weighted edges (Table 1). □

Using Lemma 5.2, all the primary outputs of circuit C can be easily identified from the graph G_C. A vertex v in G_C corresponds to a primary output of circuit C if

- the degree of vertex v is one, or
- the degree of vertex v is two and both edges incident on v have equal weights.

Lemma 5.3. *Let x_n be a primary output neuron in G_C. The threshold of x_n and the weight on an edge incident to x_n uniquely determine*

- *the gate type (i.e., AND, OR, NAND, NOR or NOT) of gate x_n*, and
- *the constants A, B and J (Table 1) associated with the neural network graph of gate x_n.*

Proof. Let I_n be the threshold of x_n and w be the weight on an edge incident to x_n. The degree of x_n is at most two (Lemma 5.2). If the degree of x_n is one, then the logic gate must be an inverter since only G_{NOT} has a primary output neuron of degree one. Therefore, from Table 1, $-2J = w$ and $J = I_n$, i.e., $J = -w/2 = I_n$. If the degree of x_n is two, then both edges incident to x_n have equal edge weights (Lemma 5.2). We have two cases:

Case 1: $w>0$. From Table 1, it can easily be verified that only in G_{AND} and G_{OR} do the edges incident on the primary output neuron have positive edge weights. Therefore, gate x_n could be an AND or OR gate. Again, from Table 1, if $w<-I_n$, then the gate is AND type with $A+B=w$ and $-(2A+B)=I_n$, i.e., $A=-(w+I_n)$, $B=2w+I_n$. If $w>-I_n$, then the gate is OR type with $A+B=w$ and $-B=I_n$, i.e., $A=w+I_n$, $B=-I_n$.

Case 2: $w<0$. The gate is NAND or NOR type. From Table 1, if $-w<I_n$, then the gate is NAND type with $-A-B=w$ and $2A+B=I_n$, i.e., $A=w+I_n$, $B=-2w-I_n$. If $-w>I_n$, then the gate is NOR type with $-A-B=w$ and $B=I_n$, i.e., $A=-w-I_n$. □

Example 5.4. As an illustration, consider the graph shown in Fig. 6. Vertex x_3 is the primary output neuron since it has two equally weighted edges incident on it. The threshold of x_3 is 7, i.e., $I_3=7$, and the weight on an edge incident to x_3 is -5, i.e., $w=-5$. Since $w<0$, gate x_3 could possibly be a NAND or NOR gate. However, $-w<I_3$ and therefore, gate x_3 is a NAND gate with $A=w+I_3=2$ and $B=-2w-I_3=3$.

Let C^n be the circuit obtained by deleting gate x_n from circuit C. Let E_{C^n} and G_{C^n} be the energy function and neural network graph, respectively, of circuit C^n.

Lemma 5.5. *G_{C^n} can be constructed from G_C in* O(1) *time.*

Proof. Using Lemma 2.2, we can determine the gate type of gate x_n and the constants A, B and J. Let E_{x_n} be the energy function of gate x_n. The proof is based on the following observation: Since C^n is obtained by deleting gate x_n from circuit C, $E_{C^n}=E_C-E_{x_n}$. Therefore, we can construct G_{C^n} from G_C by deleting vertex x_n and the edges incident on vertex x_n, and by appropriately modifying:

(1) the thresholds of neighbors of x_n, and

(2) the edge weights on the edges between the neighbors of x_n, so that G_{C^n} is the graph of the energy function E_{C^n}.

The degree of x_n is at most two (Lemma 2.1). If the degree of x_n is one, let x_q be the neighboring vertex in graph G_C and I_q be the threshold of vertex x_q. Since

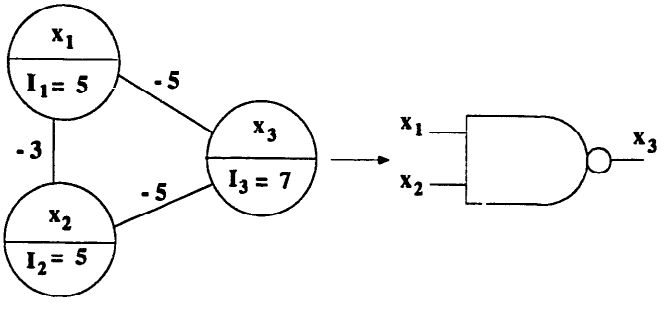

Fig. 6. A neural network graph that represents a logic gate.

circuit C'' has all signals in circuit C except signal x_n, $G_{C''} = G_C \backslash x_n$. Furthermore, removal of gate x_n changes the threshold of vertex x_q to $I_q - J$. Clearly, these modifications to G_C can be made in O(1) time.

If x_n is of degree two, let x_q and x_r be the two vertices connected to x_n, and let I_q and I_r be their respective thresholds. Let T_{qr} be the edge weight on edge (x_q, x_r) in graph G_C. $G_{C''} = G_C \backslash x_n$ and the thresholds of x_q and x_r, and the edge weight T_{qr}, are modified depending on the gate type of gate x_n. If gate x_n is an AND gate, I_q and I_r are unchanged and $T_{qr} - (-B) = T_{qr} + B$ is the new edge weight on edge (x_q, x_r). This is because, from Table 1, the AND gate x_n contributes $-B$ to the edge weight of edge (x_q, x_r) in G_C but it does not contribute to the thresholds of vertex q and r. Similarly, if gate x_n is

- OR type, I_q, I_r and T_{qr} are modified to $I_q + A$, $I_r + A$ and $T_{qr} + B$, respectively;
- NAND type, I_q, I_r and T_{qr} are modified to $I_q - A - B$, $I_r - A - B$ and $T_{qr} + B$, respectively;
- NOR type, I_q, I_r and T_{qr} are modified to $I_q - B$, $I_r - B$ and $T_{qr} + B$, respectively. □

In the sequel, the process of constructing $G_{C''}$ from G_C will be referred to as the *deletion* of gate x_n from G_C.

Given a function f and its graph G_f, we construct a combinational circuit as follows: we assume that G_f represents some combinational circuit and use Lemma 5.5 to recursively delete primary output neurons from G_f and the successive graphs obtained by deletion.

Theorem 5.6. *Let $f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{c}^T \mathbf{x}$ be a pseudo-Boolean quadratic function of n variables. Whether or not G_f is transformable to a combinational circuit can be determined in* O(n) *time.*

Proof. We rewrite f so that it is in the same form as equation (1) and let G_f be the graph associated with f. We provide a constructive proof. The construction of the combinational circuit can be performed in the following steps:

Step 1. For every vertex in G_f, compute its degree d_v and put in a list L all vertices having either (i) $d_v \leq 1$ or (ii) $d_v = 2$ and both edges incident on v having the same weight. If there exists a combinational circuit corresponding to G_f, then L is not empty (Lemma 5.2). Clearly, if there exists a combinational circuit for f, all vertices in L would be the primary outputs of the circuit.

Step 2. Choose a vertex v in L.

Case 1: $d_v = 2$. Let y and z be the vertices adjacent to v. Using Lemma 5.5, delete gate v from G_f. This appropriately modifies the thresholds of y and z and the edge weight on edge (y, z), depending on the gate type of v. If edge weight on edge (y, z) is 0, then set $d_y \leftarrow d_y - 2$, $d_z \leftarrow d_z - 2$. Otherwise, set $d_y \leftarrow d_y - 1$, $d_z \leftarrow d_z - 1$.

Include y in L if either (i) $d_y \leq 1$ or (ii) $d_y = 2$ and both edges incident on y have the same weight. Similarly, include z in L if either (i) $d_z \leq 1$ or (ii) $d_z = 2$ and both edges incident on z have the same weight.

Case 2: $d_v = 1$. Let y be the vertex adjacent to v. Using Lemma 5.5, delete gate v from G_f. This modifies the threshold of vertex y.

Set $d_y \leftarrow d_y - 1$, and if $d_y \leq 2$ we include y in L.

Case 3: $d_v = 0$. If the threshold of v is not 0, then G_f does not correspond to a combinational circuit and we stop. Otherwise mark v as a primary input of the combinational circuit.

Step 3. Set $G_f \leftarrow G_f \setminus v$ and $L \leftarrow L \setminus v$. Repeat Step 2 until L is empty.

The work needed by this algorithm is bounded by the number of edges in the graph G_f. If G_f corresponds to a combinational circuit, then G_f has $O(n)$ edges where n is the number of vertices in G_f. If G_f does not correspond to a combinational circuit, the algorithm terminates, either in Step 1 or in Case 3 of Step 2, after examining at most $O(n)$ edges. □

Note that the above procedure yields a unique combinational circuit, irrespective of how vertices are chosen in Step 2. Furthermore, our method can be viewed as a specialization of the *basic algorithm* [12] for the case when there is an ordering $x_n, x_{n-1}, \ldots, x_1$ of vertices such that x_i occurs in at most two terms not containing $x_n, x_{n-1}, \ldots, x_{i+1}$.

Example 5.7. As an illustration of Theorem 5.6, consider the quadratic 0-1 function f given by equation (4). Figure 5 shows the graph G_f associated with f. Note that G_f is not a series parallel graph. The transformation of G_f into a combinational circuit is performed as follows:

(1) Construct the list L. It consists of a single element, vertex x_5.

(2) Remove vertex x_5 from L, i.e., $L \leftarrow L \setminus x_5$.

(3) Delete gate x_5 from G_f using Lemma 5.5. From Lemma 5.3, gate x_5 is a NOR gate with $A = 1$ and $B = 2$ because $w = -3 < 0$, $I_5 = 2 < -w$ and, therefore, $A = -w - I_5 = 3 - 2 = 1$, $B = I_5 = 2$. Since gate x_5 is a NOR gate, from Lemma 5.5, the thresholds of neurons x_3 and x_4 are modified to -7 and 4, respectively. The graph $G_f \setminus \{x_5\}$ is shown in Fig. 7(a). The edge weight T_{34} is modified to 0 and therefore, edge (x_3, x_4) does not appear in Fig. 7(a). With the removal of the edge, x_3 and x_4 become degree two vertices. Since these vertices have equal weighted edges incident on them, we include them in list L. List L now has two elements, vertices x_3 and x_4.

Set $G_f \leftarrow G_f \setminus x_5$.

(4) Remove vertex x_3 from L, i.e., $L \leftarrow L \setminus x_3$.

(5) Delete gate x_3 from G_f. From Lemma 5.3, gate x_3 is an AND gate with $A = 2$ and $B = 3$. From Lemma 5.5, the thresholds of x_1 and x_2 remain unchanged and edge weight T_{12} is modified to -4. The graph $G_f \setminus \{x_3\}$ is shown in Fig. 7(b). The

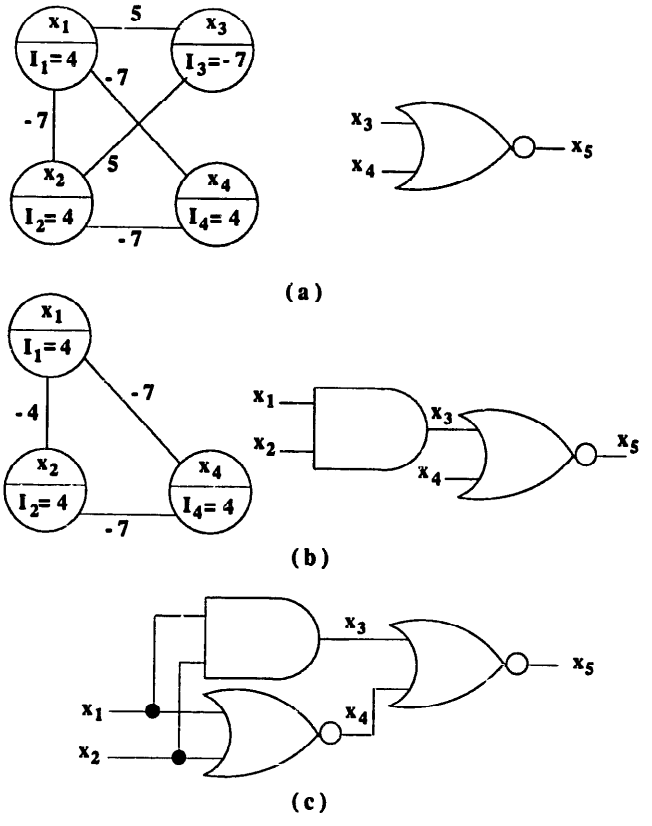

Fig. 7. Transformation of G_f into a combinational circuit.

removal of vertex x_3 leaves x_1 and x_2 as degree two vertices. However, these vertices are not included in list L since they do not have equal weighted edges incident on them. List L now has only one vertex x_4.

Set $G_f \leftarrow G_f \setminus x_3$.

(6) Remove vertex x_4 from L, i.e., $L \leftarrow L \setminus x_4$.

(7) Delete gate x_4 from G_f. From Lemma 5.3, gate x_4 is a NOR gate with $A=3$ and $B=4$. From Lemma 5.5, the thresholds of x_1 and x_2 are both modified to 0 and the edge weight T_{12} becomes 0. Therefore, the graph $G_f \setminus \{x_4\}$ consists of two isolated vertices x_1 and x_2, each with a threshold of 0. Since the degree of x_1 and x_2 is zero, we include them in list L which now has two elements, vertices x_1 and x_2.

Set $G_f \leftarrow G_f \setminus x_4$.

(8) Remove vertex x_1 from L, i.e., $L \leftarrow L \setminus x_1$.

(9) Label x_1 as a primary input of the combinational circuit since its degree is zero. L now has one element, vertex x_2.

Set $G_f \leftarrow G_f \setminus x_1$.

(10) Remove vertex x_2 from L, i.e., $L \leftarrow L \setminus x_2$.

(11) Label x_2 as a primary input of the combinational circuit.

Set $G_f \leftarrow G_f \setminus x_2$.
(12) Stop since L is empty.

The combinational circuit corresponding to G_f is shown in Fig. 7(c). Note that x_1 and x_2 are the primary inputs of the circuit.

Theorem 5.8. *The minimum of $f(x) = x^T Qx + c^T x$ can be found in linear time when the graph G_f is transformable into a combinational circuit of inverters and 2-input AND, OR, NAND, and NOR logic gates.*

Proof. From Theorem 5.6, we can determine in $O(n)$ time, where n is the number of variables in f, whether or not f can be transformed into a combinational circuit. Logic simulation of the combinational circuit for an arbitrary set of primary input signal values yields logic values for all signals in the circuit. The value of a variable in f is the logic value of the corresponding signal in the logic circuit. Therefore, the values of all the variables in f are known and an evaluation of f yields the minimum. Since the combinational circuit has n signals, logic simulation can be performed in $O(n)$ time. Therefore, a minimum of pseudo-Boolean quadratic function f can be determined in $O(n)$ time. □

Example 5.9. Consider equation (4). The logic circuit corresponding to f (Fig. 7(c)) has x_1 and x_2 as the primary inputs. Logic simulation of the circuit with $x_1 = 1$ and $x_2 = 0$ yields $x_3 = 0$, $x_4 = 0$ and $x_5 = 1$. Substitution of these values into f gives 0, the minimum of the pseudo-Boolean quadratic function f.

If the logic circuit corresponding to the function f has p primary inputs, then it has 2^p consistent labelings (Section 2). Since only a consistent labeling of the circuit corresponds to a minimum of f, the function has 2^p minimizing points.

The constructive proof of Theorem 5.6 can easily be extended to yield a polynomial time algorithm for transforming G_f into a combinational circuit that contains XOR and XNOR gates as well. Inclusion of XOR and XNOR gates in the combinational circuit will increase the class of quadratic 0-1 programs that can be formulated as logic simulation problems and, hence, solved in linear time.

If only the minimum value of f is desired and not the values of the variables at the minimum of f, logic simulation of the combinational circuit can be avoided.

Theorem 5.10. *Let $f(x) = x^T Qx + c^T x + F$, where F is the constant term. Also, let C be the combinational circuit corresponding to G_f and let K be the constant term in the energy function E_C of circuit C. The minimum of f is $F - K$.*

Proof. The procedure for transforming G_f into a logic circuit ensures that both f and E_C have the same quadratic and linear terms. Therefore, $f - F = E_C - K$. Since E_C has a minimum value 0 at all consistent labelings of signals in the logic circuit, the minimum of f is $F - K$. □

The constant K can be computed by a slight modification of the procedure for constructing a combinational circuit from G_f (see Section 5.1). Initially, set $K=0$. Whenever a subgraph in Step 2 (in the proof of Theorem 5.6) represents a gate, add the constant associated with the energy function of the gate (from Table 1) to K. When a logic circuit corresponding to G_f is completely constructed, K will be the constant term associated with the energy function of the logic circuit.

Example 5.11. Consider equation (4) in Example 5.1, where the constant $F=6$. Set $K=0$ and follow the prescribed procedure to convert G_f into a combinational circuit. When the NOR gate with $A=1$ and $B=2$ is constructed, increment K by 2 since the constant associated with the energy function of the NOR gate is 2 (Table 1). When the AND gate with $A=2$ and $B=3$ is constructed, K is unchanged since the constant associated with the energy function for the AND gate is 0. When the NOR gate with $A=3$ and $B=4$ is constructed, increment K by 4. After the termination of the procedure, $K=6$. The minimum of f is $F-K=0$.

6. Minimizing f when all elements of Q are positive

In this section, we investigate the complexity of minimizing special instances of quadratic 0-1 programming problems. As stated earlier, the problem of minimizing an arbitrary quadratic 0-1 programming problem is known to be NP-complete. However, there are three special cases when the problem is known to be polynomial time solvable. Picard and Ratliff [15] have given a polynomial time algorithm for solving the special case when all the elements of Q are nonpositive (i.e., zero or negative). Barahona [2] has given a polynomial time algorithm for the case when the graph G_f is series parallel. Crama et al. [8] generalized this result and proposed a polynomial time algorithm for the case when the graph G_f is of bounded tree width.

Here, we study the complexity of minimizing special quadratic 0-1 functions in which all the elements of Q are positive. Note that this class of functions includes the class of functions for which Theorem 5.8 applies. We show that the problem of minimizing such special functions is NP-complete.

Theorem 6.1. *Finding the minimum of* $f(x)=x^{\mathrm{T}}Qx+c^{\mathrm{T}}x$ *when all elements of* Q *are positive is* NP*-complete.*

Proof. Obviously, the problem is in NP. Hence, we need to show that some NP-complete problem is polynomially transformable to the problem of finding the minimum of a pseudo-Boolean quadratic function f with positive coefficients of all the quadratic terms. We shall transform the NP-complete problem of stuck-at fault test generation in combinational circuits [9] into the problem of finding the minimum of f.

The problem of generating a test for a stuck-at fault in an arbitrary combinational circuit can easily be transformed into a problem of generating a test for a fault in a combinational circuit of only NOR gates or NAND gates since all other logic gates can be realized as combinations of NOR or NAND gates. In [7], we have shown the construction of the neural network for an arbitrary fault in a combinational circuit. This construction can be carried out in $O(n)$ time where n is the number of signals in the combinational circuit. The neural network for a fault in a combinational circuit of only NOR or NAND gates has an energy function with positive coefficients for all quadratic terms. This is because, from Table 1, all these gates have quadratic terms with positive coefficients. Therefore, the problem of test generation in a combinational circuit is polynomially transformable into the problem of finding the minimum of the pseudo-Boolean function f with all coefficients of the quadratic terms being positive.

Since our problem is in NP and it is polynomially transformable to another known NP-complete problem (stuck-at test generation), our problem is NP-complete. □

7. Conclusion

We have presented a new class of quadratic 0-1 programming cases that can be solved in linear time. When the pseudo-Boolean quadratic function f can be transformed into a combinational logic circuit, then the minimum of f can be obtained by identifying the primary inputs of the circuit and then performing logic simulation for an arbitrary set of 0-1 values for the primary inputs. The transformation of f into a combinational logic circuit requires $O(n)$ time, where n is the number of variables in f. Logic simulation of the resultant logic circuit requires $O(n)$ time. Therefore, the minimum of f can be obtained in $O(n)$ time when f is transformable into a combinational logic circuit. We have established a direct and constructive relationship between quadratic pseudo-Boolean functions and combinational circuits.

We have also shown that the problem of finding the minimum of f in the restricted case when all elements of Q are positive is NP-complete. This result establishes an interesting connection between quadratic 0-1 programming and test generation for logic circuits.

Acknowledgement

We are grateful to Vishwani Agrawal and an anonymous referee for a careful reading of the manuscript and for numerous suggestions that have improved the clarity of the paper.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, Reading, MA, 1974).
[2] F. Barahona, A solvable case of quadratic 0-1 programming, Discrete Appl. Math. 13 (1986) 23-26.
[3] M. Boulala and J.P. Uhry, Polytope des independants d'un graphe serie-parallele, Discrete Math. 27 (1979) 225-243.
[4] M.W. Carter, The indefinite zero one quadratic problem, Discrete Appl. Math. 7 (1984) 23-44.
[5] S.T. Chakradhar, Neural network models and optimization methods for digital testing, PhD thesis, DCS-TR-269, Department of Computer Science, Rutgers University, New Brunswick, NJ (1990).
[6] S.T. Chakradhar, M.L. Bushnell and V.D. Agrawal, Automatic test pattern generation using neural networks, in: IEEE Proceedings of the International Conference on Computer-Aided Design (1988) 416-419.
[7] S.T. Chakradhar, M.L. Bushneli and V.D. Agrawal, Toward massively parallel automatic test generation, IEEE Trans. Computer-Aided Design 9 (1990) 981-994.
[8] Y. Crama, P. Hansen and B. Jaumard, The basic algorithm for pseudo-Boolean programming revisited, Tech. Rep. RRR #54-88, Rutgers Center for Operations Research (RUTCOR), Rutgers University, New Brunswick, NJ (1988).
[9] H. Fujiwara, Logic Testing and Design for Testability (MIT Press, Cambridge, MA, 1985).
[10] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).
[11] P.L. Hammer, P. Hansen and B. Simeone, Roof duality, complementation and persistence in quadratic 0-1 optimization, Math. Programming 28 (1984) 121-155.
[12] P.L. Hammer and S. Rudeanu, Boolean Methods in Operations Research (Springer, New York, 1968).
[13] P.L. Hammer and B. Simeone, Quadratic functions of binary variables, Tech. Rep. RRR #20-87, Rutgers Center for Operations Research (RUTCOR), Rutgers University, New Brunswick, NJ (1987).
[14] J.J. Hopfield, Artificial neural networks, IEEE Circuits and Devices Magazine 4 (5) (1988) 3-10.
[15] J.C. Picard and H.D. Ratliff, Minimum cuts and related problems, Networks 5 (1975) 357-370.